Software Engineering - CS16106

Name Of the Tool: JENKINS

Jenkins is an open-source automation server that facilitates building, testing, and deploying software projects. It supports the continuous integration and continuous delivery (CI/CD) principles, helping teams automate various stages of the software development lifecycle.

PURPOSE

• Compatibility Improvement:

Jenkins helps in reducing compatibility issues between different software components by implementing measures that will ensure seamless integration and operation of software across different platforms and configurations.

• Automation for Workload Reduction:

Jenkins minimizes manual workload in the testing and deployment processes to streamline repetitive tasks and increase efficiency.

• Accelerated Software Updates:

Jenkins can expedite the delivery of software updates to users, by utilizing continuous integration/continuous deployment (CI/CD) pipelines for faster and more reliable updates.

• Shared Defect Repository:

To Enhance collaboration between developers and operations in identifying and resolving defects, Jenkins
establish a shared repository accessible to both developers and operations (DevOps) for efficient tracking and resolution of software defects.

Minimum Specification required

I. Minimum hardware requirements:

• 256 MB of RAM
• 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)
Recommended hardware configuration for a small team:

- 4 GB+ of RAM
- 50 GB+ of drive space

II.Minimum software requirements:

- Java new version installed.
- Web browser: see the Web Browser Compatibility page
- For Linux operating system: Linux Support Policy

Languages Supported:

There is a diverse set of programming languages used in Jenkins, including but not limited to: Java, JavaScript, Groovy, Golang, Ruby, Shell scripts. And, since Jenkins is an automation server with hundreds of plugins, there is a huge number of technologies involved.

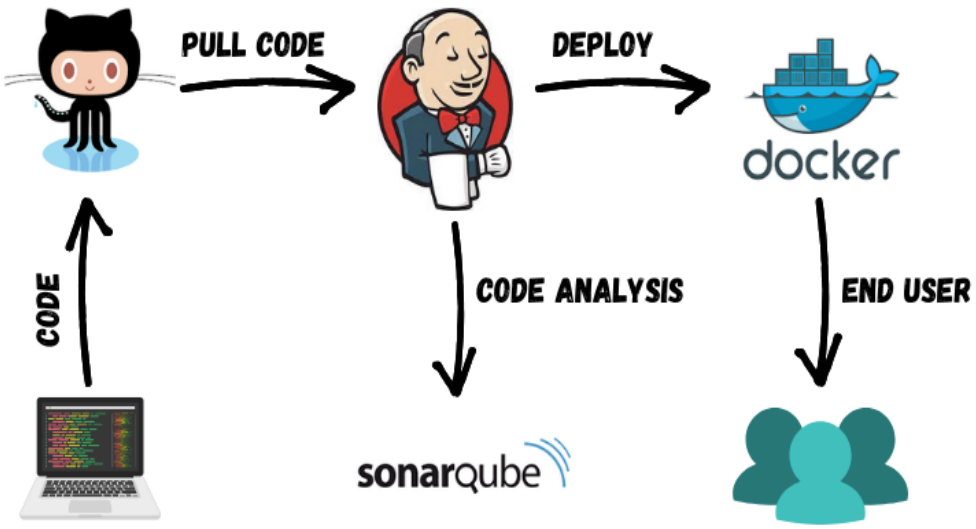
Lines Of Code:

1. Minimum Lines of Code: The minimum number of lines of code for a basic Jenkins pipeline configuration could be around 10 lines. This includes defining a pipeline with a few stages and simple steps. A basic Jenkins Pipeline configuration can indeed be quite concise, often requiring only a few lines of code to define a simple workflow.
2.
3. Maximum Lines of Code: The maximum number of lines of code for a Jenkins configuration is not bounded by a specific number. It can grow upto hundreds or even thousands of lines for complex pipelines with numerous stages, parallel execution, conditional logic, error handling, environment configurations, and extensive use of Jenkins DSL (Domain Specific Language) or Groovy scripting.

In Which Phases of SDLC Jenkins used?

Jenkins make use of pipelines to automate Building (Building phase), testing(Testing phase) and reporting on isolated changes in a larger code base in real time and facilitates the integration of disparate branches of the code into a main branch. They also rapidly detect defects in a code base, and prepare the code base for deployment (delivery), and ultimately deploy code to containers and virtual machines, as well as bare metal and cloud servers.

Actions Performed



- So, we have created a CI/CD pipeline in which we have pushed the code to GitHub and then pulled the code from it to Jenkins by creating pipeline.
- Then we have tested the code on sonarqube, which is a static code analysis tool in which we can scan the bugs and vulnerabilities of the code.
- After scanning the code we have deployed it on docker.
- We have done everything while using the AWS EC2 instance.
- We have used three different instances-:

one for Jenkins, second for SonarQube and third for deployment in docker, but we have used the same SSH key for it.

- After that, we installed Jenkins from jenkins.io on our AWS ec2 instance.
- Meanwhile, we worked on allowing port 8080 so that we can access it on our browser. We did this by clicking on the instance id of Jenkins then selecting Security groups and after scrolling down we clicked on edit inbound rules where we added the rule.
- After that using the public Ip address of Jenkins, we ran the port 8080(<public IP address>:8080) After that we created our user id along with installing suggested plugins.

Then the next work we did was to push the code from GitHub to the Jenkins' server.
For that we created freestyle project pipeline from the “New item” option where we gave a name to our pipeline.

- Then we made changes in the Configure in Jenkins by providing GitHub repository URL in the source code management.
- For build trigger type, we chose GitHub hook trigger for GITScm polling. It will trigger the build automatically whenever we make changes in the repository. In order to use this, we added a webhook to our GitHub repository.

After that, we clicked on “Build now” to build our code. On a successful build, we

checked that on making any commits to GitHub was it doing an automated build which it did successfully.

After that, in order to test our project, we created a server for our SonarQube.

- We downloaded it from sonarsource.com then we unzipped the file and ran sudo apt install unzip in the terminal and inside Linux folder ran ./sonar.sh.
- Meanwhile, we worked on allowing port 9000 so that we can access it on our browser. We did this by clicking on the instance ID of sonarque, then selecting Security groups and after scrolling down security we clicked on edit inbound rules where we added the rule. After that, using the public Ip address of Sonarqube, we ran the port 9000(<public IP address>:9000).
- Then we created the project manually and provided the main branch name. Then we for CI method we clicked on Jenkins.
- For Devops platform, we clicked on: GitHub
- To Create the jenkinsfile:we will click on other.
- Now, after that we generated tokens that will be authenticated for our particular project to test our project.

Meanwhile, we installed SonarQube plugin and SSH2 Easy plugin in Jenkins.
After that we will build our code again and scanning will start in SonarQube.

Once our code got passed, we deployed it. Now for deployment we installed docker from docker.com.

- After that, in configure of Jenkins we added docker-server within server group centre where we provided the IP of docker server.
- After that, in run shell, we ran the command touch test.txt, through which we got test.txt file. Then we created a new file named Docker file in our GitHub repository.
- Basically, after that, we copied the contents of the directory to the remote server.
- we created docker container to run our port on 8085.

On Building again we were successfully able to run our website on port 8085.

Comparison With Other Tools-

- Bamboo, provides structured approach to CI/CD, offers a more polished and integrated experience within the Atlassian ecosystem, with a focus on ease of use and seamless integration with other Atlassian tools. Jenkins, on the other hand, provides unparalleled flexibility and customization options, making it suitable for a wide range of projects and use cases, together with potentially higher setup and configuration provided.
- GitLab CI/CD a part of GitLab platform, which offers an integrated solution with version control, issue tracking, and other DevOps features. Unlike Jenkins, GitLab CI/CD is hosted, meaning users don't need to manage their own infrastructure.
- TeamCity like Jenkins is a self-hosted CI/CD server along with build,

deployment. It is known for its ease of use and comprehensive feature set, with advanced build queue management capabilities. Unlike Jenkins, TeamCity is a commercial product with a licensing fee based on the number of build agents.

INPUT

For Jenkin's users we must provide some of the inputs at different stages like:

- Provide a Source code on which we do experiments.
- Set up the configurations.
- Install the necessary plugins.
- Authentication and Authorization is needed to use the resources by Jenkins.

Experimental Reference:

- For Jenkins:jenkins.io
- For SonarQube: sonarsource.com
- For docker:docker.com

STATUS - Jenkin is in working condition.

User can log in using their username and password, they can see the build status at any time. Commit history shown to the users who have collaborated.

Team Grp-11:

1. Ayati Sonkar (20214300)
2. Dilpreet Kaur (20214031)
3. Dipti Kumari (20214105)
4. Ishika Agrawal (20214204)
5. Kiran Kumari (20214106)